SCIENTIFIC and TECHNOLOGICAL COOPERATION

between

RTD ORGANISATIONS in GREECE

and RTD ORGANISATIONS in U.S.A, CANADA, AUSTRALIA, NEW ZEALAND, JAPAN, SOUTH KOREA, TAIWAN, MALAISIA and SINGAPORE

HELLENIC REPUBLIC

MINISTRY OF DEVELOPMENT

GENERAL SECRETARIAT FOR RESEARCH & TECHNOLOGY

International S & T Cooperation Directorate, Bilateral Relations Division

Project MILTIADES: **M**ult**I-L**ayer **T**echn**I**ques for **A**ttack **DE**tection **S**ystems

Deliverable 1.1: Requirements Analysis

This document presents an overview of the project's objectives and the results of the first phase of the project. In this phase, the requirements of the proposed system were collected and analysed. In particular, in this document, we elaborate on the functionality, performance, as well as the deployment requirements.

Due Delivery Date	30/09/2006
Actual Delivery Date	31/01/2006
Participants	Virtual Trip, FORTH-ICS

Project Partners

FORTH-ICS	Project Leader	Greece
Columbia University	Project Leader	US
Virtual Trip	Co-operating organisation	Greece

Table of Contents

1. Motivation

Over the last few years we have been witnessing an ever-increasing amount of computer attacks on the Internet. These attacks, which in the colorful language of computers are called computer viruses, computer worms, or simply Internet epidemics, have demonstrated that they can compromise a very large number of computers within just a few minutes. For example, in January 2003, the Sapphire worm compromised more than 50,000 computers in less than 30 minutes, and was the first worm to experimentally demonstrate that such a rapid spread of an Internet epidemic is possible. To make matters worse, recently, controlled lab experiments have demonstrated that sophisticated computer worms can compromise tens of thousands of computers in a matter of seconds [SMPW2004]. These Internet epidemics have recently resulted in millions of hours in computer downtime, in disabling of banking ATM systems, and in the grounding of airplane flights[1]. Worms have already demonstrated that they can penetrate critical infrastructures such as nuclear power plants.

Given that worms have demonstrated that they can disable banks, ground flights, and penetrate nuclear power plants, we should start thinking what would happen if worms start crippling the critical infrastructures on top of which we base our lives, including electric power plants, water supplies, fossil fuel tanks, and telecommunication systems.

[1] http://www.cnn.com/2003/TECH/internet/01/25/internet.attack/

2. Problem statement

Worms are able to rapidly propagate on the Internet because they are self-replicating programs. That is, computer worms replicate on their own: once released on the Internet they do not need our help, or intervention, in order to replicate. As long as there exist vulnerable servers on the Internet, a computer worm can propagate by preying on them, compromising and devouring one server at-a-time.

In order to compromise, replicate, and propagate on the Internet, all computer worms use more-or-less the following procedure:

1. The worm finds a vulnerable server on the Internet
2. The worm sends to the vulnerable server a specially crafted network packet which, when received and acted upon by the server
    a. It triggers a bug in the vulnerable server's code,
    b. It alters the server's execution path, and
    c. It forces the server to start executing code provided (or controlled) by the worm.
3. Once the vulnerable server is compromised, it is then used as a stepping stone in order to replicate the worm to other vulnerable servers, starting the cyber from step one above all over again.

By indefinitely repeating the above three steps, a worm can potentially compromise all the vulnerable servers on the Internet, using each one of them as a stepping stone in the process. As long as there exists even a single computer which runs the vulnerable software, the worm can still infect the vulnerable computer and propagate. This implies that simplistic procedures such as, "rebooting the infected computer" and/or "removing the worm" from the infected machine, are not sound approaches to eradicate the worm from the cyberspace. The only way to eradicate the worm once and for all, would be to completely remove from the Internet all servers running copies of vulnerable software.

Several researchers have already started working on developing methods to identify, mitigate, and respond to Internet epidemics as soon as they emerge, and hopefully before they manage to compromise (nearly all) vulnerable servers.

2.1 Research Questions

Some of the research approaches are based on the fact that traditionally all instances of a computer worm are identical to each other. Indeed, as we have already said, a worm is a self-replicating program, that is, a program which replicates (i.e. creates copies of) itself. All these copies are usually identical to each other, or at least have very large parts in common. Based on the observation that different replicas of the same worm are identical, or at least share large common substrings, computer researchers have developed systems which identify packets belonging to worms during a worm outbreak, partly, based on the overwhelming similarities which exist between different replicas of a computer worm [AAM2005, SEVS2004, VSGB2004]. These methods examine all packets in the network in order to find frequently occurring substrings, which have not been seen in the past (at least in such quantities) belonging to network packets going from several sources to several destinations. If they find such substrings, then these may probably be part of a new worm.

To evade detection based on similarity patterns, polymorphic worms encrypt their payload (i.e. their body) with a different key each time they replicate themselves. Therefore, all "replicas" of a polymorphic worm on the Internet will look almost entirely different from each other, because their payloads will be encrypted with different keys. However, in order to decrypt their malicious payload, polymorphic worms are forced to prefix their encrypted body with a non-encrypted decryptor. As soon as the worm gains the flow of control of a remote computer, its first step must be to start executing its (non-encrypted) decryptor which then decrypts the worm's encrypted body. Obviously, these non-encrypted decryptors are usually the Achilles heel of polymorphic worms.

To evade detection through the identification of the non-encrypted identical decryptors, worm writers have developed methods which metamorphize the code of the decryptor, a line of technical work, which gave birth to what is now known as metamorphic worms. Each instance of a metamorphic worm obfuscates its original instructions by using two interesting approaches: First, a metamorphic worm substitutes some of its original instructions with equivalent (sequences of) instructions, and second, a metamorphic worm blends its original instructions by adding to its payload other, often irrelevant, instructions. For example, if the code of the worm contains an instruction to multiply a register by the number two, a metamorphic worm may achieve the same result (i.e. multiply a register by two) by using the equivalent instruction of adding the register to itself. To obfuscate its real instructions, a metamorphic worm may also intersperse within its code several instructions that operate on registers and/or memory addresses which are not actually needed by the worm.

Armed with payload encryption and instruction obfuscation, polymorphic and metamorphic worms are much more difficult to detect than their original worm counterparts, and have given worm writers a significant head-start compared to security scientists. To make matters worse, today there exists little, if any, research which can be used to develop systems that are able to detect this new generation of polymorphic and metamorphic worms.

2.2 State of The Art

Cyber-attacks, as are currently being exemplified by worms and viruses, are a relatively recent phenomenon. It wasn't however, until the CodeRed worm outbreak of 2001 that the research community started to realize the real threat of Internet worms and to start working in order to advance the research about Internet epidemics [MSB2002]. Based partially on experimental evidence provided by the Code Red worm, Staniford et al.

predicted that even faster worms would follow, and were actually proven true by the subsequent release of the Sapphire-Slammer worm in early 2002 [MPS+2003], which infected more than 70,000 computers in less than 30 minutes leaving systems administrations little, or better yet, not at all, time to react.

In the last few years, researchers have started to work on worm detection and worm fingerprinting methods. Early worm detection methods were based on network telescopes which by monitoring a large range of unused IP addresses, they were able to observe (obviously unsuccessful) connection attempts to those IP addresses, which were usually related to some form of a computer attack [MVS2001]. Network telescopes are sometimes complemented by honeypots, i.e. idle computers which neither have any ordinary users, nor do they provide any advertised services to the community [LL+2003]. Since they provide no obvious service, honeypots should neither receive not generate any traffic. If honeypots receive traffic, then this is usually the result of an attacker trying to compromise the honeypot. If they also start to generate traffic, then this is probably an indication that the honeypot has been compromised and is used as a stepping stone to compromise more computers. After detecting a worm through port scanning or honeypots, it is usually necessary to fingerprint it, that is, to create a signature of the worm: a machine readable description which can be used by firewalls and/or Intrusion Prevention Systems to detect and block the worm.

Although these methods are effective with detecting traditional worms, their effectiveness is questionable in the case of polymorphic and metamorphic worms. There have been some attempts to extend them in the case of polymorphic worms the replicas of which still have some overlaps [NKS2005], but the effectiveness of such approaches is obviously limited when faced with poly- and meta-morphic worms whose replicates do not have any common substrings.

In another attempt to improve the performance and accuracy of Network-based Intrusion Detection Systems (NIDSes), Dreger et al. have proposed to use a hybrid system where NIDSes communicate with Host-based Intrusion Detection Systems (HIDSes) which

provide information that is not easily available at the network-level [DKPS05]. This information includes decryption of network-level encrypted data, and protocol processing results.

Finally, to prevent sophisticated attacks hidden in the payload of seemingly innocent packets, Microsoft has developed the Shield system which, based on a set of signatures (called shields) is able to detect and discard attack packets [WGSZ2004].

3. Objectives

The objectives of this project are:

- To contribute towards the development of a second-generation early warning system, which will be able to detect and fingerprint polymorphic and metamorphic worms.

- To contribute towards the containment of zero-day polymorphic worms by designing and developing appropriate mechanisms which will impede the spreading rate of such worms by poisoning their intelligence gathering services with stale information

- To design and develop a novel defence mechanism, the Application Level Intrusion detection System (ALIS) which will explore pioneering ways to detect polymorphic and metamorphic worms which can not be otherwise identified at the network level.

4 Methodology and Research Directions

The following planned work will be a step towards covering the ground lost to worm writers, by designing, implementing, and deploying, methods to detect and mitigate polymorphic and metamorphic worms.

Recent research efforts indicate that there is no “silver bullet”/single solution to the worm problem. It is the purpose of this project to explore techniques that are fundamentally different in their principles, and determine how they interact and complement each other towards addressing the problem of computer worms. Our contributions revolve around the following two research directions:

- IP Address Randomization

- Application-Level Intrusion Detection Systems (ALISes) and Abstract payload execution

4.1 IP address Randomization

To evade detection during their outbreak, and generate as little traffic as possible, sophisticated worms gather information about their targets several weeks before they launch their attack. During those weeks they probe a very large number of IP addresses, if possible all 4 billions of them available in the Internet today (IP version 4), in order to find those hosts which are vulnerable to the planned attack. All vulnerable hosts found, are included in a special list of targets, which is frequently referred to as the “hit-list”. Instead of attacking computers at random during the outbreak, as naive worms do, sophisticated hit-list-based worms only attack computers included in the hitlist and therefore (1) they generate the minimum traffic possible, and (2) they do not generate any

unsuccessful (TCP) Internet connections. Therefore, hit-list-based worms propagate at the maximum possible speed, evading their timely detection by worm detection systems.

To slow down the rate of their spread, and if possible neutralize those hit-list worms, we propose to conduct research towards mechanisms which will make the contents of the hit list stale. Armed with a stale hit list, a worm will not only not be able to successfully infect hosts, but it will also make several unsuccessful attempts to compromise non-vulnerable computers, resulting in several unsuccessful TCP connections. These unsuccessful connections will probably be more visible to firewalls and intrusion detection systems, which will quickly take notice of the spreading worm. To put it simply: a stale hit-list will slow down the spreading worm, and make it visible to firewalls and Intrusion Detection Systems.

4.2 Application-Level Intrusion Detection Systems

To protect themselves against malicious intruders, organizations usually employ Network-level Intrusion Detection Systems (NIDSes) in their gateways to the Internet. NIDSes, inspect all incoming traffic against a preloaded set of known attack signatures (i.e. attack rules) in order to see whether any network packet(s) match any of the attack signatures. As soon as the NIDSes find network packets which match any one of the attack signatures, they log the offending packets and alert the system administrators of the intrusion attempt.

Although IDSes have been successfully used to identify and prevent traditional attacks, they are getting increasingly less effective when faced with the next generation of polymorphic and metamorphic worms for several reasons:

- Traditional Intrusion Detection Systems executing on the network gateway usually lack the context in which the incoming network packets will be received and acted upon. Therefore, even though they may identify suspicious packets which may contain polymorphic and metamorphic worms, NIDSes are not able to

decide whether these packets will trigger an attack when they will be received by the final destination

- Polymorphic worms, as we have already explained, encrypt their payload, and therefore, are difficult to detect. To obfuscate their presence even further, polymorphic worms may even make their decryptor dependent on the context (e.g. the memory contents) of the receiving application. That is, before starting decoding, the decryptor may read the contents of a memory location and use them in the decoding process. Since the context of the application within which the decryptor will run (i.e. the value of the mentioned memory location) is not known at the network level, decrypting the polymorphic worm at network level is very difficult, if not impossible.

To overcome the above limitations of the traditional NIDSes, we explore the usefulness of Application-Level Intrusion Detection Systems (ALISes), which have several advantages compared to NIDSes:

- Access to address space information. ALISes run inside the address space of the end-user application and therefore they have access to the appropriate context needed to discover what will happen when the network packets are received by the end-user application and are acted upon
- Access to larger computing capacity. By running at several different host computers, the aggregate computing capacity available to all ALISes of an organization, in total, will probably exceed the computing capacity available to a single NIDS. Therefore, ALISes may be able to perform more sophisticated (i.e. time consuming) algorithms in the incoming network packets.
- Access to system-call sequences. Since ALISes run inside the address space of an application, the information available to them is not restricted to network packets (as the information available to network-level IDSes is restricted to), but it may also include system call sequences, and memory contents. This implies that ALISes may be more accurate than NIDSes in their search for Internet epidemics.

- Richer Signatures. Since the information available to ALISes may be far richer than that available to NIDSes, the signatures that can be implemented by ALISes may be significantly more elaborate than those of NIDSes. Such signatures may include statements about the system calls made by the end user application, the memory contents of the end application, the data received by the end user application, and so on. On the contrary, the signatures that can be implemented by NIDSes are usually restricted to the headers and payload of network packets.
- More accurate zero-day worm detection. By having access to a wider variety of more accurate information, ALISes may be able to detect zero-day cyberattacks which evade traditional network-level systems. Such cyberattacks may include encrypted (polymorphic) worms, and metamorphic worms.

5 Requirement Analysis

This section lists generic intrusion detection system requirements as well as functionality, deployment and performance requirements for a platform implementing a second-generation early warning system. This platform will be built with the aim to detect and fingerprint polymorphic and metamorphic worms. As stated above, we specifically investigate the applicability of two mechanisms. Firstly, a mechanism that will impede the spreading rate of such worms by poisoning their intelligence gathering services with stale information. Secondly, a mechanism that will detect at the application level polymorphic and metamorphic worms, which can not be otherwise identified at the network level.

5.1 Generic Requirements

In the bibliography, the following generic requirements are defined for a good intrusion detection system:

1. A system must recognize any suspect activity or triggering event that could potentially be an attack.
2. Escalating behavior on the part of an intruder should be detected at the earliest stage possible.
3. Components on various hosts must communicate with each other regarding level of alert and intrusions detected.
4. The system must respond appropriately to changing levels of alertness.
5. The detection system must have some manual control mechanisms to allow administrators to control various functions and alert levels of the system.
6. The system must be able to adapt to changing methods of attack.
7. The system must be able to handle multiple concurrent attacks.
8. The system must be scalable and easily expandable as the network changes.
9. The system must be resistant to compromise, able to protect itself from intrusion.

10. The system must be efficient and reliable.
11. The operation of the system should be transparent and straightforward.
12. The system should not be detectable by an attacker.
13. The system should correlate data from multiple sources to identify suspicious activity and patterns of vulnerability and exploitation.
14. Analysis capability should be able to operate on historical data
15. The system should be able to operate in real or near-real time to assess the current state of the network.
16. Detect intrusions specific to a designated area of protection.
17. Monitoring and scanning systems should have no noticeable effect on normal network operations.
18. Policy-enforcing systems should cause minimal degradation of normal network service.
19. Systems should impose no significant load on local area network (LAN) or wide area network (WAN) bandwidth
20. The system should be engineered to allow easy integration of new functionality and capability as threats, technologies, and techniques evolve.

5.2 IP Address Randomization

Initially, investigate different hitlist generation strategies and focus on their effectiveness in terms of natural decay rates. An example of a hitlist generation strategy is one where a distributed application provide protocol functionality for crawling that can be exploited by an attacker to rapidly build hitlists.

Then, consider some basic hitlist characteristics, such as the speed at which a hitlist can be constructed, the rate at which addresses already change (without any form of randomization), and how address space is allocated and utilized. This will be achieved by performing measurements on the Internet.

Research in what form the proposed technique, Network Address Space Randomization (NASR), is acceptable in practice to intentionally accelerate hitlist decay. Hitlists tend to decay naturally for various reasons, as hosts disconnect and applications are abnormally terminated. A rapidly decaying hitlist is likely to decrease the spread rate of a worm. It may also increase the number of unsuccessful connections it initiates and may thus increase the exposure of the worm to scan-detection methods.

Study the interaction between NASR and other defense mechanisms in more depth. As NASR is likely to at least slow down worms, it may provide the critical amount of time needed for distributed detectors to kick in, and for reactive approaches to deploy patches or short-term filters. Determining whether this is indeed a possibility requires further experimentation and analysis.

Investigate ways to avoid assigning an address to a host that has significant overlap in services (and potential vulnerabilities) with hosts that recently used the same address. For instance, randomization between hosts with different operating systems, e.g., between a Windows and a Linux platform appears as a reasonable strategy.

Based on simulations, decide the scope of a NASR implementation. Apparently, this technique is more affective if the scope is restricted scope to more local regions (e.g., the subnet-level). In general, candidate network segments appear to be those that already perform dynamic address allocation, such as DHCP pools for broadband connections, laptop subnets, wireless networks. It is pointless to implement NASR behind NATs, as the internal addresses have no global significance. It is sufficient to change the address of the NAT endpoint (e.g., DSL/home router) to protect the internal hosts.

DNS would need to be dynamically updated when hosts change addresses. Implementing NASR requires the DNS name to accurately reflect the current IP address of the host. This means that the DNS time-to-live timers need to be set low enough so that remote clients and name servers do not cache stale data when an address is changed. The NASR mechanism also needs to interact with the DNS server to keep the address records up to

date. It is reasonable to ask whether this could increase the load on the DNS system, given that lower TTLs will negatively affect DNS caching performance. As expected, defending against hitlists that are generated very fast requires more frequent address changes.

Some nodes cannot change addresses and those that can may not be able to do so as frequently as NASR would want. The reason for this is that addresses have first-class transport and application level semantics. For instance, DNS server and routers. Generally, all active TCP connections on a host that changes its address would be killed, unless connection migration techniques are used. Such techniques are not widely deployed yet and it is unrealistic to expect that they will be deployed in time to be usable for the purposes of NASR. Many applications are not designed to tolerate connection failures. Fortunately, many applications are designed to deal with occasional connectivity loss by automatically reconnecting and recovering from failure.

Assess the “collateral damage” caused by NASR. The damage depends on how frequently the address changes occur, whether hosts have active connections that are terminated and whether the applications can recover from the transient connectivity problems caused by an address change.

Estimate the typical subnet utilization levels and observe whether NASR continues to be effective in slowing down the worm, when deployed in a subset of the network. The worm might still infect the non-NASR subnets quite rapidly, but with a slowdown caused by the worm failing to infect NASR subnets.

Experiment with IPv4 addresses, as deployed today, and project NASR effectiveness in an Ipv6 network. In an IPv6 Internet, the address space is so much bigger that randomization could be even more effective.

Assess the constraints that limit the applicability of the proposed approach, such as the administrative overhead for managing address changes, services that require static

addresses and applications that cannot tolerate address changes or suffer performance-wise when addresses change frequently.

5.3 Application-Level Intrusion Detection Systems

Design, implement and evaluate detection heuristics that test byte sequences in network traffic for properties similar to polymorphism. Speculatively execute potential instruction sequences and compare their execution profile against the behavior observed to be inherent to polymorphic shellcodes.

Protect effectively against attacks (i.e. generate effective signatures to stop most of the attacks). Protected applications should be able to withstand repetitive attacks at a much higher rate than that of unprotected applications. That means better immunity to denial-of-service attacks.

Generate fast attack signatures, in the order of ten milliseconds time, with just a single sample. Speedy signature generation makes it possible to distribute and deploy these signatures in the Internet to stop fast-spreading worms.

Low overheads under normal operation. Measure application's server CPU usage as well as throughput and latency, with and without ALIS. Calculate the number of attacks per second required to degrade a protected server's performance to a certain level (e.g. 50% availability) and compare with the capacity of an unprotected server under the same type of attack.

No false positives. For all the attacks evaluated, minimise or zero any false positives. Manually analyse application code to verify the correctness and accuracy of the generated signatures.

The method should be applicable to COTS software, without access to source. The approach should not require any modifications of the protected server software, or access to its source code.

Evaluate ALIS against “real-world” attacks, i.e. use existing exploit code against popular applications.

Identify limitations of the proposed method.

6 Conclusions

This document presented an overview of the project's objectives and a list of requirements the proposed system will have to meet.

The main objective is the development of a second-generation early warning system, which will be able to detect and fingerprint polymorphic and metamorphic worms.

This objective will be met by a) designing and developing appropriate mechanisms which will impede the spreading rate of such worms by poisoning their intelligence gathering services with stale information and b) designing and developing a defence mechanism which will explore ways to detect such worms at the application level because they can not be otherwise identified at the network level.

Finally, a list of functionality, deployment and performance requirements was presented. The two proposed mechanisms have to meet a number of requirements to effectively serve their purpose.

One open question that deserves further research and analysis is how worm creators would react to the widespread deployment of NASR and ALIS. There is a real possibility that worm creators could come up with other measures to counter those defenses.

7 References

[AAM2005] P. Akritidis, Kostas Anagnostakis, and E.P. Markatos: **Efficient Content-Based Fingerprinting of Zero-Day Worms**. *Proceedings of the International Conference on Communications (ICC 2005)*, Seoul, Korea, 16-20 May 2005

[DKPS05] H. Dreger, C. Kreibich, V. Paxson and R. Sommer. **Enhancing the Accuracy of Network-based Intrusion Detection with Host-based Context**. In *Proceedings of DIMVA* 2005.

[LL+2003] J. Levin, R. LaBella, H. Owen, D. Contis, and B. Culver. **The Use of Honeynets to Detect Exploited Systems Across Large Enterprise Networks**. In *Proceedings of the 2003 IEEE Workshop on Information Assurance*. June 2003.

[MPS+2003] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. **The Spread of the Sapphire/Slammer Worm**. *IEEE Security and Privacy*, 1(4), July 2003.

[MSB2002] D. Moore, C. Shannon, and J. Brown. **Code-Red: a case study on the spread and victims of an Internet worm**. In *Proceedings of the 2002 Internet Measurement Workshop*. http://citeseer.ist.psu.edu/moore02codered.html

[MVS2001] D. Moore, G. M. Voelker, and S. Savage. **Inferring Internet Denial-of-Service Activity**. In *Proceedings of the USENIX Security Symposium*, Aug. 2001.

[NKS2005] J. Newsome, B. Karp, and D. Song. **Polygraph: Automatically Generating Signatures for Polymorphic Worms**. In the *Proceedings of the IEEE Symposium on Security and Privacy* (Oakland 2005), Oakland, CA, May, 2005

[SMPW2004] . Staniford, D. Moore, V. Paxson and N. Weaver. **The Top Speed of Flash Worms**, In *Proc. ACM CCS WORM*, October 2004.

[SEVS2004] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. **Automated Worm Fingerpriting**. *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*. 2004

[VSGB2004] S. Venkataraman, D. Song, P.B. Gibbons, and A. Blum. **New Streaming Algorithms for Fast Detection of Superspreaders**. May 2004, IRP-TR-04-23

[WGSZ2004] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier. **Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits**. ACM SIGCOMM, August, 2004.